Programming 2
Tutorial and Practical 3

This week's tutorial and practical sessions are about designing and implementing a non-trivial scenario using object oriented techniques and practices.

The Scenario

A courier company needs to develop a system that tracks the maintenance schedule of its delivery fleet. This is to be performed by keeping usage and maintenance related details about each vehicle in its fleet, so that service intervals are kept and unserviceable vehicles are not accidentally rostered on duty.

When a vehicle is to take a job, information about the job (the vehicle to be used, and the distance in kilometres) is provided to the system. If the vehicle is past its service interval, the job should be refused. If still in a usable condition, the appropriate vehicle is updated with the usage information, and it should return with an approximate wear-and-tear cost of the particular job. When a vehicle is serviced, the system is notified.

The Vehicles

The company uses both vans and trucks. Both vehicle types have a registration number (used to identify the vehicle); information such as the year, make and model; and maintenance information (its current odometer reading, the reading at its last service, and the service interval). Service intervals are specific to each individual vehicle, and cannot be modified once the initial information is entered.

Trucks have an additional piece of information; its load capacity.

The wear and tear expenses for a job can be calculated as such:

- Vans: a flat rate of 60 cents per kilometre
- Trucks: a rate of 50 cents per tonne (load capacity), per kilometre *(for example: a truck with a load capacity of 2.5t, has a rate of $1.25 a kilometre.)*

When a vehicle is serviced, the service interval resets. For example; if a new van with a 20,000km service interval is first serviced when the odometer is at 14,449km, it should be able to work until its odometer first exceeds 34,449km; at which point it must be serviced before it can be used again.

The Manager

At the top level, the maintenance manager section of the system must allow *at least* the following core actions to take place:

- Add a vehicle to the system
- Display information about a vehicle
- Provide job information for a vehicle (at which case it is either refused' or is accepted, vehicle data updated, and returns with a wear-and-tear estimate)
- Service a vehicle

You may find that other functionality may be necessary, to help facilitate these actions.

Tutorial Session

Work out a feasible object design to model this system, and consider ways in which the various core actions can be implemented.

Points to consider: *(not an exhaustive list)*

- How to represent (and relate, where appropriate) the entities in the system
- What information is required by the system, and when
- How vehicles will store and manipulate their service information
- How to represent failure/job refusal, compared to success/wear-and-tear data?
- How the wear-and-tear calculations can be implemented most elegantly

Now, draw your attention to how the system can be tested, to ensure that all of the critical functions are working properly.

Consider the important functions that need testing. Which are they?

- Will each function work the same way, in all situations?
- If not, what are those different behaviours, and under which circumstances should they occur?
- How can those circumstances be simulated? What test data is required?

Practical Session

Implement the above system using the class design developed in the tutorial. If you experience difficulties with the implementation, discuss with your peers (and the teaching staff) the possible ways of overcoming the problems.

Instead of writing a complete menu-driven user interface for the maintenance manager, save yourself some time and simply hard-code some test cases as explored in the tutorial class. (Since the test regimen has already been worked out, it is not necessary to provide full interactivity – all we are interested in is, "*does the system work?*")